

CIRCULATION COPY
BUREAU OF RECORDS
MAY 1 1980


UCRL-83791 Rev. 1
PREPRINT

VERIFICATION OF TIMING CONSTRAINTS ON LARGE DIGITAL SYSTEMS

Thomas M. McWilliams

This paper was prepared for submittal to the
Seventeenth Design Automation Conference,
Minneapolis, Minnesota, June 1980

April 7, 1980



Lawrence
Livermore
Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement recommendation, or favoring of the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Verification of Timing Constraints on Large Digital Systems

Thomas M. McWilliams

Lawrence Livermore Laboratory
University of California
and
Computer Science Department
Stanford University

Abstract

A new approach to the verification of the timing constraints on large digital systems has been developed. The associated algorithm is computationally very efficient, and provides early and continuous feedback about the timing aspects of synchronous sequential circuits as they are designed. It also provides means for conveniently verifying the design in sections, permitting the section-by-section timing verification of designs which are too large to examine as a unit on existing computer systems. A system using this algorithm has been implemented, and has been used to verify the timing constraints on the design of the S-1 Mark IIA processor.

1 INTRODUCTION

In order that a digital system perform correctly, a designer must take into account the possible propagation delays associated with each of the elements making up the system. If a path through a digital system has either too long or too short a delay associated with it, then the value of the circuit may be wrong at a critical point, causing the circuit to calculate an incorrect result. This is called a timing error. Digital logic as it is currently implemented is intrinsically susceptible to such errors, and their complete elimination from all portions of a digital logic system is essential to a realistic guarantee that the logic will perform reliably and reproducibly under all variations in data and programs. This paper addresses the early and efficient detection of these timing errors, so that digital logic designers can henceforth frequently examine their designs for them as the design proceeds, thereby finding timing errors before the design progresses so far that the errors become difficult to eliminate.

A system which uses these ideas has been implemented and is called the SCALD Timing Verifier. It inputs the design of a synchronous sequential system given in the SCALD Hardware Description Language [6], and analyzes it, comprehensively searching it for timing errors. SCALD (Structured Computer-Aided Logic Design) is a complete computer-aided design system which inputs a graphics-based, hierarchical description of a digital logic design, and thereupon generates a complete set of low-level documentation which includes that necessary to implement it in hardware [6,7]. The Timing Verifier performs a complete timing constraint verification based on the minimum and maximum propagation delays of the circuit components, their set-up and hold times, minimum pulse width constraints, and wire delays.

One of the principal features of the Timing Verifier is its ability to verify designs by modules. This not only permits its use on computers with limited memory size, but also allows timing constraints to be checked as a design progresses, even on a day-by-day basis. This is particularly important in that it allows timing errors to be corrected before they have a chance to propagate their effects throughout the design, or to cause major changes to be required late in the design. It also supports an accurate estimation of the cycle time of a digital logic machine before its design is completed.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

2 PREVIOUS APPROACHES

There have been a number of previous approaches to the verification of timing constraints in digital systems; these can be grouped into two main categories: logic simulation [3,4,5,10] and worst-case path analysis [2,12].

The logic simulation approach poses several problems. It requires either a complete design including any microcode and diagnostics, or some way of generating patterns to drive the undefined signals. Waiting until the design is completed to start simulation presents the problem that bugs are not found until late in the design cycle. Generating patterns to drive undefined signals is very time-consuming and difficult, especially when the patterns need to go through the "worst-case" set of states. Simulation is also a very inefficient way of finding timing errors, because of the need to run through a large number of states in order to test all of the "worst-case" timing paths. In fact, for most large digital systems, it is impossible to have a high degree of confidence that the worst-case states have all been tested, and both the human and computational efforts required to do simulation are orders of magnitude greater than those required by the Timing Verifier.

The worst-case path analysis approach examines all paths through the combinational logic between registers or latches, searching for the longest and shortest paths. This approach only works on fairly simple combinational logic, and tends to fall down on complex synchronous sequential circuits. It is also computationally expensive, but does provide feedback to the designer early during the design cycle, without the need to generate detailed test patterns.

In contrast, the SCALD Timing Verifier eliminates most of the problems associated with these approaches, allowing the design's timing properties to be verified as the design proceeds, without the need to generate complex test patterns. It also uses a computationally efficient algorithm to cover all of the states needed to identify and quantitate all timing errors. Handling circuits in which logic simulation of portions of the circuit is needed to understand the detailed timing is another of its capabilities.

3 THE SCALD TIMING VERIFIER

The SCALD Timing Verifier operates on synchronous sequential systems, and checks all of the logic-level timing errors which occur within those systems. These include the non-satisfaction of the set-up, hold, or minimum pulse width time requirements for registers, latches, and other complex functions. In addition to these errors, it checks the timing on control signals which are ANDed with clock signals to verify that they are stable while the clock is asserted, in order to avoid any possible hazard conditions on control-conditioned clock lines. The Timing Verifier takes into account both the minimum and maximum propagation delays of all of the system's components, including the interconnections between them.

3.1 THEORY OF OPERATION

Within synchronous sequential circuits, most signals can be changing only during particular parts of the clock period. For example, it may be possible for a particular signal to be changing only during the second half of the clock cycle, given that all of the components making up the system are within their timing specifications.

Consider a register which can be clocked only at a particular time within the clock period. The output of the register can change only during a short time after it is clocked, so that it is guaranteed to be stable for the entire clock period except around the point at which it is clocked. The output of a gate driven from this register can then be changing only during an interval of time determined by its propagation delay and that of its connection to the register and the time when the output of the register is changing.

Determining when a given signal may be changing and when it is stable within the clock period is the key step in the operation of the Timing Verifier. Once this has been accomplished, it is relatively easy to check all of the timing constraints placed on the circuit. For instance, in order to check the set-up and hold times on a register, all the Timing Verifier has to do is to see if its input could be changing at a time when it might be clocked.

If the timing of the circuit never depended on the values of signals, but only on when they were changing or stable, the Timing Verifier would be relatively simple. Clock signals have a value which is periodic, and have the same value every cycle, so they are easy to manage. The signals which are difficult to treat are those whose values affect the circuit timing, and which have different values during different clock cycles. For example, a control signal which determines whether a register is clocked during a given cycle affects whether the output of the register might change that cycle. If the circuit depends on the register not changing every cycle, then the Timing Verifier must do *case analysis* in order to avoid generating false error messages. It is therefore required to check the timing both when the control signal is true, and when it is false. This is potentially a very time-consuming process. In practice it has turned out not to be so, because most signals have a "worst-case" state. For example, the worst-case for most registers is to assume that they are clocked every cycle. Only for those situations in which both the clocked and unclocked cases need to be checked separately must the Timing Verifier examine both of them. In such cases, the Timing Verifier remembers the values of all the signals which are not affected by the signal which is having case analysis done on it. It then needs to recompute only the signals affected by the case analysis.

For a given circuit, the Timing Verifier has some number of cases to analyze. Which signals require case analysis and what cases need to be evaluated for these signals are specified by the designer. This might appear to be a possibly quite inefficient process, but it has been found in the design of the S-1 Mark IIA processor, that very few cases need to be so analyzed in practice.

The basic procedure followed by the Timing Verifier in doing case analysis is to take the *first* case specified by the designer, and to calculate for each signal in the system when it could be changing during the clock cycle. Once it has done this, it checks for possible violation of timing constraints for that case. It then goes on to the *next* case specified by the designer to be checked, recomputing only those signals which are *different* from the *previous* case, and checking for any possible timing errors in that case. Continuing this process, it checks *all* of the cases, thereby performing a complete check of the circuit for timing constraint violations.

3.2 CIRCUIT CLOCK PERIOD

Circuits being verified must contain one basic clock, whose period is specified to the Timing Verifier. If different parts of the circuit being verified run at different clock rates, then the period specified to the Timing Verifier is the least common multiple of the different clock periods. For example, a processor might have an instruction unit which has a period of 30 nsec and an execution unit which has a period of 15 nsec. In this case, the period specified to the Timing Verifier would be 30 nsec. Clock signals which occur within the circuit may occur at any phase within the basic clock period.

3.3 CIRCUIT MODEL FEATURES

Circuits are described to the Timing Verifier in terms of gates, registers, latches, set-up and hold time constraints, and minimum pulse width constraints. More complex functions are then defined in terms of these primitives, through the use of graphics-based macros, using the SCALD Hardware Description Language [6,7].

The following sections define the values which are used to represent the behavior of signals. They include the definitions of the primitive components with which the design is specified to the Timing Verifier.

3.3.1 Value System Used To Represent Signals

At any instant in time, every signal in the circuit being timing-verified has exactly one of seven values, with the following associated meanings:

Value	Meaning
0	false, or 0
1	true, or 1
S or STABLE	signal is stable, not changing
C or CHANGE	signal may be changing
R or RISE	signal is going from zero to one
F or FALL	signal is going from one to zero
U or UNKNOWN	initial value used for all signals

The value of a signal over the clock period is represented by a linked list, each node of which specifies a signal value and the time duration of that value. The sum of the durations of all the nodes in the list must exactly equal the period of the circuit being analyzed.

When a signal propagates through a gate or wire where it is delayed by a variable amount of time, then *skew* is added to the signal representation, denoting the uncertainty in when the signal will subsequently change. This skew is maintained separately in the signal representation to preserve information about the width of pulses. This is done to avoid incorrect assertions by the Timing Verifier that minimum pulse width requirements have not been met. If two or more changing signals are combined, the skew of the resulting signal cannot be represented separately. It is therefore incorporated into the signal representation by using the CHANGE, RISE, and FALL values.

3.3.2 Definition of Combinational Functions

This section defines the basic combinational functions used by the Timing Verifier. All other combinational functions may then be defined in terms of these basic functions.

The following tables define the INCLUSIVE-OR (OR), AND, EXCLUSIVE-OR (XOR), CHANGE (CHG), and NOT functions for the seven-value logic system used in the Timing Verifier.

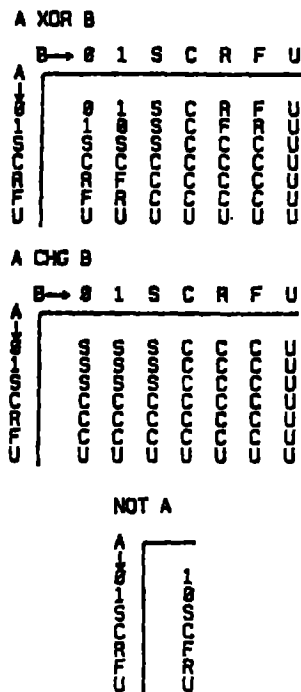
These functions are uniformly defined to give worst-case values. For example, when the signal values "STABLE" and "RISING" are OR'ed together, the resultant signal value given is "RISING". This is because the output in this case will either be stable or a rising edge, and the rising edge is the worst-case value.

A OR B

B →		0	1	S	C	R	F	U
A ↓	0	0	1	1	1	1	1	1
	1	1	1	1	1	1	1	1
	S	1	1	S	C	R	F	U
	C	1	1	S	C	R	F	U
	R	1	1	S	C	R	F	U
	F	1	1	S	C	R	F	U
	U	1	1	S	C	R	F	U

A AND B

B →		0	1	S	C	R	F	U
A ↓	0	0	0	0	0	0	0	0
	1	0	1	1	1	1	1	1
	S	0	1	S	C	R	F	U
	C	0	1	S	C	R	F	U
	R	0	1	S	C	R	F	U
	F	0	1	S	C	R	F	U
	U	0	1	S	C	R	F	U



The output of the "CHANGE" function has the value "UNDEFINED" if any of its inputs are undefined. If all of its inputs are defined, then it has the value "CHANGE" if any of its inputs are changing; otherwise it has the value "STABLE". It is a useful function in modeling complex combinational logic, where the actual function being performed is not significant to the verification process. Common examples are in the modeling of parity trees and adders, in which cases the Timing Verifier cares only when the outputs of these circuits are changing, not about their actual values. This results in a great reduction in the complexity of adequately modeling these functions.

3.3.3 Definition of Registers and Latches

The Timing Verifier has two models for registers which are shown in Figure 3-1. The first register model just has "CLOCK" and "DATA" inputs, and can change its output only on the rising-edge of its "CLOCK" input. The output of the register will be set to the "CHANGE" state during the time following the rising-edge of "CLOCK" as determined by the minimum and maximum delays of the register. Unless the "DATA" input is a true or false during the rising-edge of the "CLOCK" input, the output will be set to the "STABLE" value for the rest of the cycle; otherwise, it will be set to the value of the "DATA" input. The example in Figure 3-1 shows a minimum delay of 1.0 nsec and a maximum delay of 3.8 nsec being specified for the register, which is 32-bits wide.

The second register shown in Figure 3-1 is the same as the first, except that it has asynchronous "SET" and "RESET" inputs in addition to the "DATA" and "CLOCK" inputs. The minimum and maximum propagation delays from all of the inputs are the same, and are given by the delay property of the register. If chips with different propagation delays from different inputs are to be modeled, then buffers are used on the various inputs to insert the proper delays. Primitives with different delays from different inputs could be implemented to improve execution efficiency, if desired.

The Timing Verifier has two models for latches, as shown in Figure 3-2. The "OUTPUT" of the first latch follows the "DATA" input when the "ENABLE" input is high, and holds the last value given by the "DATA" input when the "ENABLE" input is low. The "SET" and "RESET" inputs on the second latch in Figure 3-2 operate the same as for the register, and override the operation of the latch when they are non-zero. The minimum and maximum propagation delay from all of the inputs on the latch are the same, and is given by the "DELAY" property. For the example shown in the Figure, the minimum propagation delay is 1.0 nsec, and the maximum propagation delay is 3.5 nsec.

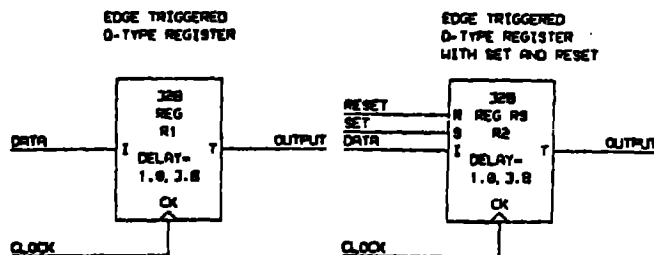


Figure 3-1
Two register models used by Timing Verifier

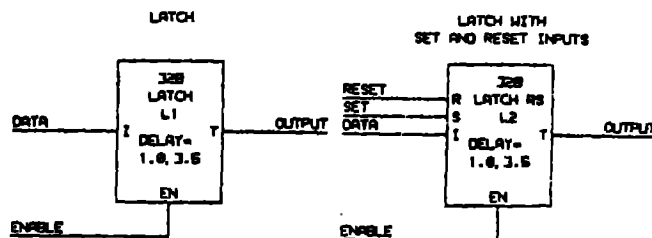


Figure 3-2
Two latch models used by Timing Verifier

3.3.4 Setup and Hold Time Checkers

There are two primitive functions, shown in Figure 3-3, which are used to check set-up and hold times. The first checker is called a "SETUP HOLD CHK", and checks to see that the signal connected to the "I" input is stable for a period around the rising edge of the "CK" input. The "SETUP" property specifies the set-up time interval, which is the length of time the input signal must be stable before the rising edge of the clock input. The "HOLD" property specifies the hold time interval. This is the length of time the input signal must be stable after the rising edge of the clock input.

The second primitive shown in Figure 3-3 is a "SETUP RISE HOLD FALL CHK" primitive. It checks the set-up time interval of the input before the rising edge of the clock input, and the hold time interval after the falling edge of the clock input. It also checks to see that the input "I" is stable for the entire time interval over which the clock input "CK" is true. This type of set-up and hold checker is needed to verify the timing constraints on components such as memory chips.

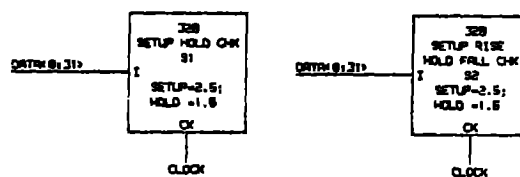


Figure 3-3
Set-up and hold time checkers used by Timing Verifier

3.3.5 Minimum Pulse Width Checkers

The minimum pulse width checker is used to specify verification of minimum pulse width constraints. Clock inputs to components typically have a minimum pulse width requirement which says that when they go high, they must stay high for some specified interval of time, and that when they go low, they must stay low for some specified time interval. Figure 3-4 shows the "MIN PULSE WIDTH" primitive, which shows a minimum *high* pulse width of 5.0 nsec being specified, as well as a minimum *low* pulse width of 3.0 nsec.



Figure 3-4
Minimum pulse width checker

3.4 SIGNAL ASSERTIONS

In order to be able to analyze partially designed circuits, the Verifier must have timing assertions on as-yet undefined signals. Undefined signals with no assertions are taken to be always stable, to prevent them from giving rise to numerous spurious timing error messages.

Two types of assertions are used for specifying clocks, and one is used for defining the behavior of control and data signals.

3.4.1 Clock Assertions

There are two categories of clock signals: precision and non-precision. The only difference between precision and non-precision clock specifications is the default skew used by the Timing Verifier when none is explicitly given by the designer. Skew is generated by the variation in the interconnection delay to the different parts of a large digital system and by the variations in delay between the different buffers used in the clock generation. In the design of a large digital system, these variations can become quite large, and may degrade performance unacceptably. To reduce such skew to within acceptable limits, the shorter clock paths can have additional delays deliberately inserted into them. Because the delays in a clock distribution system may vary between successive implementations of a design, in many cases it must be adjusted by hand, using some type of adjustable delay for each of the clock lines. By use of this technique, the skew can be reduced to below some designer-specified value. In order to verify the timing in a design which has been so de-skewed, it is necessary to describe in detail how the clocks will be adjusted within the design specification. A number of features have been provided to make this task as easy as possible, and will be described in the section on evaluation directives.

If a clock signal is adjusted to some specified skew, then an assertion can be given within its signal name signifying that fact. Assertions are given at the end of signal names and are preceded by a period. They are considered part of the signal name by the rest of the SCALD system, which thereby guarantees that all of the assertions for a given signal are consistent by definition.

The format for the assertions for the precision and non-precision clocks are

```
<precision clock> ::= <signal name> .P <assert spec>
<non-precision clock> ::= <signal name> .C <assert spec>
<assert spec> ::= <value specification>
                  <skew specification> <polarity assertion>
<value specification> ::= <time range> |
                        <time range> , <value specification>
<time range> ::= <time> | <time> - <time>
                  | <time> + <time>
<time> ::= <real number>
<skew specification> ::= | (<minus skew>, <plus skew>)
<minus skew> ::= <negative real or zero>
<plus skew> ::= <positive real or zero>
<polarity assertion> ::= | L
```

The time units in which the clocks are specified are normally some fraction of the cycle time. For example, one eighth of the cycle is the basic clock interval used in the design of the S-1 Mark IIA processor, which is discussed in greater detail subsequently. Specifying clock intervals as fractions of the cycle time (rather than in absolute time units) has the basic advantage of allowing the relative timing within the design to be scaled automatically if the cycle time is changed.

An example of clock specification is

XYZ .C4-6 L

which states that the clock signal goes from high to low at time 4, and from low to high at time 6. The signal

XYZ .C2-3,5-6

is high from 2 to 3 and from 5 to 6, and is low for the rest of the clock cycle. If a single time is given instead of a range, a time interval of one clock unit is assumed. For example,

XYZ .C2,5

is equivalent to the previous signal. The signal

XYZ .P2,5

is again equivalent, except that it is a precision clock, which means that it has a different default skew. In general, it was found in the design of the S-1 Mark IIA processor that having two types of clocks — those that have been adjusted to reduce skew, and those that haven't — was convenient. The motivation was to only adjust those clocks which must be adjusted, in order to reduce cost.

If a plus sign is given between the two time variables instead of the minus sign, then the second number specifies a width in nanoseconds, rather than the time of the end of the pulse in clock units. This allows widths of clocks that don't scale with the cycle-time of the circuit to be specified. For example,

XYZ .P2F10

specifies a clock that goes high at clock unit time 2, and stays high for 10.0 nsec.

3.4.2 Stable Assertions

The stable assertion is used to specify when a control or data signal is stable, and when it may be changing. Its general form is

<signal name> .S <value specification> <polarity assertion>

For example, the name XYZ .S4-8 says that the signal is stable from time 4 to time 8, and may be changing during the rest of the cycle.

This type of assertion has several uses. First, it allows the designer to specify his assumptions about when signals are valid (i.e., not changing) as he creates them in the design process, and those assumptions will be used by the Timing Verifier until the signals are generated by hardware. For signals so generated, the designer's initial timing assertion is checked against the timing of the actual, generated signal, and an error message is output if the assertion is

violated. In the design of the S-1 Mark IIA processor, most signal names have stable assertions in them. This greatly improves the readability of the design, since a signal name explicitly includes a specification of when it is valid.

Putting these "stable" assertions on interface signals is the key to the ability to verify a design in sections. After each section is verified, SCALD checks to see that all interface signals have the same timing assertions in the sections which they connect together. If no such section has a timing error and if all of the interface signals of all sections have consistent assertions on them, then the entire design must be free of timing errors.

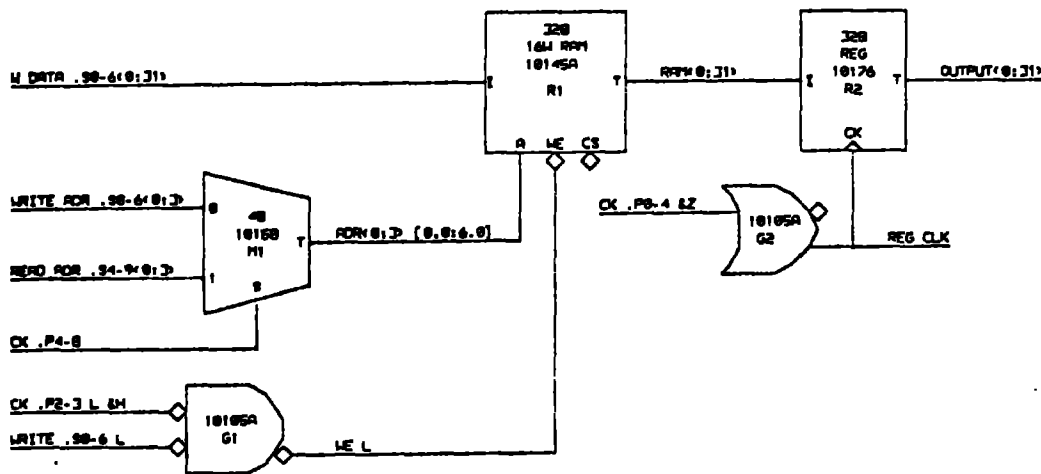


Figure 3-5
Example macro definition

3.4.3 Interconnection Delay Specification

Taking into account the effects of interconnection delays throughout the design process is essential if maximum system performance is to be attained when the design is completed. The consideration of these delays needs to be approached from two different points of view, depending on whether or not the design is far enough along to allow the actual interconnection delays to be calculated. If the interconnection delays can be calculated from detailed simulation of the transmission line properties of the interconnections, then these delay values are used by the Timing Verifier when checking timing constraints within the design. If the interconnection delays are not yet known, the Timing Verifier uses a default interconnection delay for each signal. If the designer wishes, he can specify within the design a range for the interconnection delay for a specific signal, which will then override the default specification.

3.5 EVALUATION DIRECTIVES

Evaluation directives are used to specify:

- That the control signals being ANDed with a given clock signal must be stable while the clock is asserted. This is used to detect possible hazards that could be generated on the output of a gate, resulting in false clocking of the circuit which the gate controls. In addition, these directives cause the Timing Verifier to assume that the control signals will be enabling the gate, so that its output value will be determined only from the value behavior of the clock signal.
- The tuning of clocks in systems which have hand-adjusted clocks to reduce skew. Additional information is needed here since the prints don't specify how the clocks are adjusted.

Consider the circuit shown in Figure 3-5. The clock signal "CK.P2-3 L" is being ANDed to the control signal "WRITE.S0-6 L" to generate a write-enable pulse for the RAM array. The "&H" directive specifies checking that the control signal "WRITE.S0-6 L" is stable during the interval over which the clock is asserted, to ensure that the "write" will be either solidly enabled or completely disabled. In addition, this directive says the timing specified by the clock signal is to be adjusted so that it refers to the time at which the *output*, rather than the *input*, of the gate changes. This corresponds to a circuit in which the clock signals are adjusted to eliminate the skew generated by gating of the clock lines. The last thing that the "&H" directive does is to tell the Timing Verifier to ignore the value of the "WRITE.S0-6 L" signal, allowing the clock signal to always propagate through the gate.

There are a number of different directives of the same general type as the "&H" directive. For example, the "&Z" directive on the signal "CK.P0-4 L" states that the clock timing refers to the time at which the *output* of the gate changes. If multiple directives are given after a signal, such as "&HZ", then the first letter refers to the first level of gating after the directive, the second refers to the second level of gating, etc. There is no limit on the length of a directive string.

3.6 CASE ANALYSIS

Sometimes the timing analysis of a circuit requires that a number of different cases within the circuit need to be analyzed, one after another, in order to check their timing properties. This requirement occurs because the timing of the circuit is a complex function of the values of some signals, which cannot be analysed when combined using the "STABLE" and "CHANGING" states.

Figure 3-6 gives a circuit example which needs case analysis.

If the circuit is analysed without case analysis, where the signal "CONTROL SIGNAL" has the value "STABLE", then the delay from the signal "INPUT" to the signal "OUTPUT" would be calculated to be 40 nsec. The problem is that the Timing Verifier would be unable to determine that both of the multiplexers could not select the "1" input at the same time. To use case analysis, the designer would specify that the signal "CONTROL SIGNAL" needs to be analysed separately for the cases when it is true and when it is false. For the first case, the Timing Verifier would then set the signal "CONTROL SIGNAL" to the value "0" whenever the circuit would normally set it to the value "STABLE". For the next case, it would set it to the value "1" whenever the circuit would normally set it to the value "STABLE". In this way, the two select lines on the multiplexers would always be set to complementary values, and the delay from the signal "INPUT" to the signal "OUTPUT" would be calculated to be 30 nsec for both cases.

3.6.1 Case Specification

The designer must identify and specify those signals which need to be handled by case analysis. He does this by creating a text file specifying the cases that need to be evaluated. This text file explicitly states which signals need to have their "STABLE" states mapped into either "0" or "1" values ("FALSE" or "TRUE"). Consider the following specification:

```
X=1,Y=0,Z=*;
X=0,Y=0,K=*,Z=*;
```

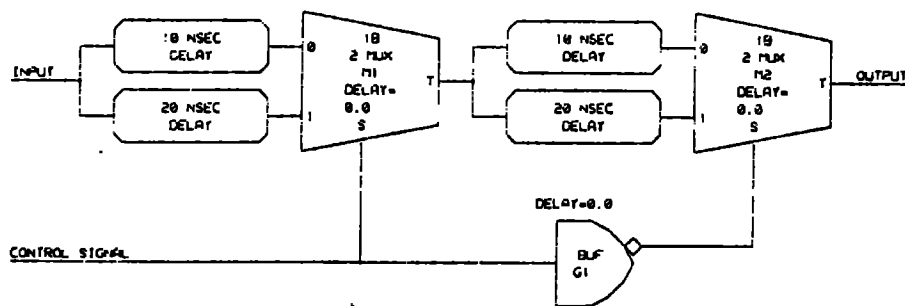


Figure 3-6
Example of circuit requiring case analysis

This specification specifies 6 cases to be evaluated. The text before the first semicolon specifies two cases: The first with "X=1,Y=0,Z=0" and the second with "X=1,Y=0,Z=1". The construct "SIGNAL=*" says to consider the cases of that signal being a "0" and a "1". The semicolon delimits one set of case analysis specifications from the next. The second line of this example specifies 4 cases to be examined. They are "X=0,Y=0,K=0,Z=0", "X=0,Y=0,K=0,Z=1", "X=0,Y=0,K=1,Z=0", and "X=0,Y=0,K=1,Z=1".

3.7 TECHNIQUE USED FOR CIRCUIT EVALUATION

The first step in evaluating a circuit is to initialize to "UNDEFINED" all signals without assertions. Signals with clock assertions are set to the value specified. Signals with stable assertions are set to the value "STABLE" during the time specified by the assertion, and to the value "CHANGING" the rest of the time. Signals which are specified in the case analysis file will be set to the value specified for the case being calculated whenever they otherwise would be given the value "STABLE".

In the next step, the Timing Verifier evaluates all of the primitives which define the circuit by looking at their current input values and based on these calculating new output values. Whenever a new output value is different from its old value, all of the primitives that are driven by that output are added to a list of primitives to be evaluated during the next pass of the Timing Verifier. This process continues, reevaluating those primitives which have had their inputs changed, until all of the signals stop changing. At that point, the Timing Verifier knows the value of each signal over the clock period, for the first case to be analyzed.

The next step is to evaluate all of the set-up and hold times, and minimum pulse width checkers, based on the value of their inputs, and to output error messages reporting any errors detected. This error checking includes set-up and hold time constraints specified both by the set-up and hold time primitives and by the "&A" and "&H" evaluation directives.

At this point, the first case has been evaluated, and the Timing Verifier is ready to evaluate the next case. This involves changing the values of those signals specified by the case analysis file, and reevaluating those primitives whose inputs are affected. This process is continued, as in the first case, until all signals stop changing. At that point, the second case has been checked. The Timing Verifier will continue this process, incrementally reevaluating the network, until all of the cases specified by the user have been checked.

4 CIRCUIT VERIFICATION EXAMPLE

Figure 3-5 shows an circuit example to be analysed by the Timing Verifier. This circuit consists of a 16-word by 32-bit

register file, a 32-bit output register, a 2-input multiplexer which selects between the read and write addresses for the register file, and several gates. The circuit is designed to run with a cycle time of 50 nsec. The default wire delay used by the Timing Verifier in checking this circuit was 0.0 to 2.0 nsec, and the default clock skew for the clocks was -1.0 to +1.0 nsec. The time unit used in the specification of the clocks and assertions is 6.25 nsec, which gives 8 clock units per cycle.

One of the most useful features of the Timing Verifier is its ability to analyse all of the timing properties of a design as the design proceeds, rather than having to wait to be used until the design is completed. As such, it can input the description of this circuit example, which would typically be a small section of a much larger system, and determine if it contains any timing errors.

The stable assertions on the input signals which are not generated in this circuit are crucial to the ability to verify a design in pieces. For example, the assertion on the signal "W DATA S0-6" states that it is stable from time 0 to time 6, and that it may be changing during the rest of the cycle, i.e. from time 6 to time 8. The assertion on the signal "READ ADR S4-9" says that it is stable from time 4 to time 9, and may be changing the rest of the cycle, i.e. from time 1 to time 4. This may seem a little strange at first, but the cycle time of the circuit is 8 clock units long, and the assertion specification is taken to be modulo the cycle time.

Considering Interconnection delays on incomplete designs presents some interesting problems. If the actual wire delays are known for the signals in the circuit, they can be used in the analysis. If not, the Timing Verifier will use a default wire delay, unless the designer specifies wire delays for specific signals. The default wire delay of 0.0 to 2.0 nsec was used for all of the wires in this example, except for the address lines on the register file, where the designer specified that it could be anywhere from 0.0 to 6.0 nsec.

Figure 4-1 exhibits the summary output listing generated by the Timing Verifier, showing the values of the signals over the cycle time of the circuit. For example, the first entry says that the address lines "ADR<0:3>" are stable at the beginning of the cycle, and start changing 0.5 nsec into the cycle. They then go stable 5.5 nsec into the cycle, and stay stable until 25.5 nsec into the cycle. They are then changing from 25.5 nsec to 30.5 nsec, after which point they stay stable for the rest of the cycle.

Figure 4-2 contains the set-up and hold time errors which were detected by the Timing Verifier. The first message states that the "SETUP HOLD CHK" primitive specified a set-up time interval of 3.5 nsec, followed by a hold time of 1.0 nsec, and that the set-up time was violated. The next two lines give the values seen by the "SETUP HOLD CHK" primitive on the data and clock inputs. They show the data not becoming stable until 11.5 nsec into the cycle, and the clock starting to rise 11.5 nsec into the cycle. Thus, the set-up time interval specified was missed by the full 3.5 nsec. The next error message shows that the set-up time interval on the output register was violated. The data didn't go stable until 47.5 nsec into the cycle and the clock starting rising at 49.0 nsec, missing the specified set-up time interval of 2.5 nsec by 1.0 nsec.

Values of all signals

ADDR(0:3)	S:0.0, C:0.5, S:5.5, C:25.5, S:30.5	
CK .P0-4	R:0.0, I:1.0, P:24.0, 0:20.0, R:40.0	(constant value)
CK .P2-3	0:0.0, R:11.5, I:13.5, P:17.0, 0:19.0	(constant value)
CK .P4-6	P:0.0, 0:1.0, R:24.0, I:20.0, P:40.0	(constant value)
OUTPUT(0:31)	S:0.0, C:0.5, S:7.5	
RAM(0:31)	S:0.0, C:5.0, S:20.5, C:30.0, S:45.5	
READ ADDR .50-0(0:3)	S:0.0, C:6.5, S:25.0	
READ CLK	R:0.0, I:1.0, P:24.0, 0:20.0, R:40.0	
W DATA .50-0(0:31)	S:0.0, C:37.5	
WE	0:0.0, R:11.5, I:13.5, P:17.0, 0:19.0	
WRITE .50-6	S:0.0, C:37.5	
WRITE ADDR .50-6(0:3)	S:0.0, C:37.5	

Figure 4-1
Timing Verifier output showing values of signals

Setup, Hold and Minimum Pulse Width errors

Setup time error: Setup Time = 3.5, Hold Time = 1.0		0:0.0, R:11.5, I:15.5, P:17.0, 0:21.0
CK INPUT = WE	(0.0)	S:0.0, C:0.5, S:11.5, C:25.5, S:30.5
DATA INPUT = ADDR	(0.0)	
Setup time error: Setup Time = 2.5, Hold Time = 1.5		R:0.0, I:3.0, P:24.0, 0:20.0, R:40.0
CK INPUT = READ CLK	(0.0)	S:0.0, C:5.0, S:22.5, C:30.0, S:47.5
DATA INPUT = RAM	(0.0)	

Figure 4-2
Set-up and hold time errors found by Timing Verifier

5 REPRESENTATION OF SIGNAL VALUES

The Timing Verifier represents in memory the value of each signal over the circuit cycle time. It uses a linked list to do this, which has the format shown in Figure 5-1. For each signal, there is a "VALUE BASE" record with a free storage link, a field to store the skew, a pointer to the evaluation string, and a pointer to the linked list representing the signal value. The "VALUE" record specifies the signal value and the width of that value. The sum of all of the "VALUE WIDTH" fields on the linked list is required to exactly equal the cycle time of the circuit being verified.

The "SKEW" field is used to represent skew caused by delaying the signal by a variable amount of time. Consider the example in Figure 5-2. The gate has a minimum delay of 5.0 nsec and a maximum delay of 10.0 nsec. The two input signals will be ORed together as if the gate had zero delay, and the value of the output signal will then be delayed by the minimum delay. The skew field will then be set to the difference between the maximum and the minimum delay of the gate. By doing this, rather than by using "RISING" and "FALLING" values to represent the uncertainty in when the signal will transition between a zero and a one, the symmetry information about the width of pulses is preserved since the rising and trailing edges of the signal are delayed by the same amount. When modeling a technology in which the rising and falling values of signals are different, this algorithm will have to be modified to take this asymmetry into account.

This separate representation of skew can be used in essentially any situation in which a signal value is merely being delayed by a variable amount. However, if two signals are being combined, then the skew of the combined value in general cannot be simply represented with a single field. Because of this, when two signals are combined, their skew is inserted into the resultant signal representation using the "RISING" and "FALLING" values. For example, the output signal "Z" from the last example is shown in Figure 5-3 with its skew inserted into the signal value.

The "EVAL STR PTR" field is used to keep track of the evaluation string associated with the signal value. For example, if the evaluation string "HZZW" is given on the input of a gate, then each letter specifies how to evaluate a subsequent level of gating. Each gate will remove the letter which specifies how to evaluate it, and will pass along the rest of the string and the output value from the gate, in order to specify how to evaluate the next level of gating. The string "HZZW" specifies the evaluation of 4 levels of gating, with the "H" controlling the first level, and the "W" controlling the last level.

6 VERIFICATION OF THE S-1 MARK IIA

The SCALD Timing Verifier has been used in the design of a high performance processor, the S-1 Mark IIA [9]. This use has served to validate the great utility of the endre present approach to timing verification, and has also provided interesting performance statistics on the Timing Verifier operation. The Mark IIA is a highly pipelined processor which is designed to usually issue a new instruction every 50 nsec. The machine has a vector instruction unit which is designed to process vector operands at a pipelined rate of 25 nsec.

The Timing Verifier has been used during much of the past year, on a daily basis, to check the design for timing behavior as it has progressed toward implementation. The approach taken has been to work on the design for about a day, and then to enter the new design into the SCALD system, via the Stanford University Drawing System (SUDS). The design is then processed through the SCALD Macro Expander, which checks the design for syntax errors and generates a file which represents the expanded design. The execution statistics for running the Macro Expander on the S-1 Mark I system (which has a throughput rate approximately equivalent to an IBM 970/168) for a portion of the Mark IIA design consisting of 6357 MSI ECL-10K and ECL-100K chips is shown in Table 6-1; this portion contains 97,709 2-input gates-equivalent of logic and 1,803,136 bits of high speed memory. These execution statistics are broken down into three parts. The first part is the time required to read the input files and build the data structures to represent the design. Next, the Macro Expander does an expansion of the design to generate a summary listing, and builds up a data structure which resolves all synonyms between different signals (Pass 1). Finally, the Macro Expander expands the design again, this time outputting the fully elaborated design for use by the Timing Verifier or the SCALD Layout Program (Pass 2).

MACRO EXPANSION EXECUTION STATISTICS	Time, minutes
Reading input files and building data structures	1.92
Pass 1 of macro expansion	8.42
Pass 2 of macro expansion	6.18
	16.52

TIMING VERIFIER EXECUTION STATISTICS	
Reading input files and building data structures	4.45
Generating cross reference listings	0.72
Verifying circuit	6.75
Generating timing summary listing	0.22
	12.14
Total for both Timing Verifier and Macro Expander:	28.66

Table 6-1
Execution statistics for 6357 chip example

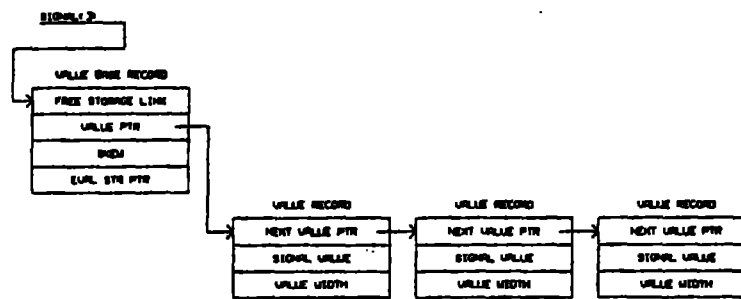


Figure 5-1
Data structures used to represent signal values

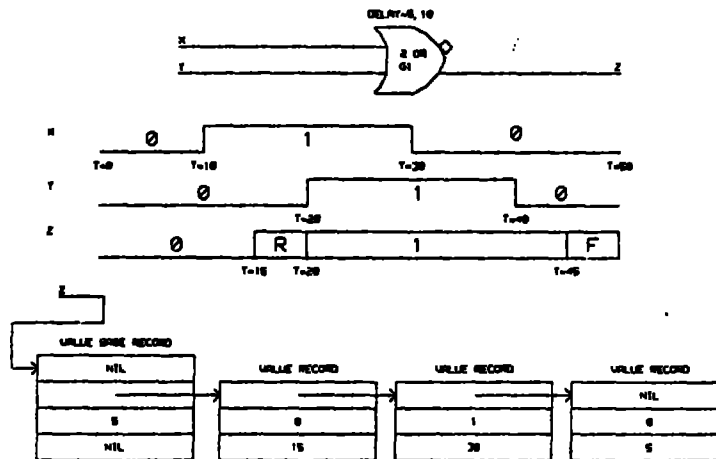


Figure 5-2
Example showing how skew is handled

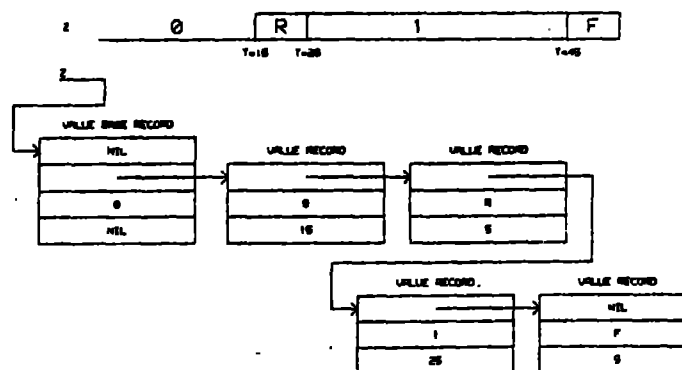


Figure 5-3
Output signal Z with skew represented in signal value

The Timing Verifier takes 4.45 minutes reading in the output from the SCALD Macro Expander, and then building up its data structures. It then generates some cross reference listings, which aid the designer in finding where signals are generated and

where they are used within the design. The next step is the Timing Verification. This takes 6.75 minutes, or about 49 milliseconds per primitive. In doing this verification, 20,052 events were processed,

where an event is caused by an output being given a new value, which in turn causes all primitives which use that output value to be updated. An event then took 20 milliseconds to process. This verification was for a single-case verification.

The amount of time required to analyze a case is proportional to the number of events which must be processed for that case. In general, only those signals which are affected by the case analysis need to be recalculated. The Mark IIA processor is a pipelined processor, in which every pipeline stage must take the same amount of time to execute; it was therefore found that case analysis is only rarely required. In fact, case analysis was found to confer so little advantage in the Mark IIA design process that it wasn't implemented until near its end. However, for some design styles, for instance those in which variable length cycles are used, case analysis capability is very important.

Table 6-2 gives the storage required for data structures used during the Timing Verification. Representing the circuit description is the single largest part of this requirement, requiring 37.8% of the 5.7 megabytes of total memory used. The circuit description is comprised of a characterization of each primitive used, with a description of which signals were passed to each of its parameters. This is the main data structure used while the circuit is being verified, and averages in size to 260 bytes per primitive. The PASCAL compiler used doesn't pack its records, so all fields require four bytes, except characters and booleans, which take one byte.

The next largest part of the storage requirement goes to the storing of signal values. A linked list is stored for each signal in the system representing its value. For the current example, there were

33,152 of these value lists stored, each of which had a base record followed by an average of 2.97 value records. The average amount of memory needed to store the value of a given signal was then 56 bytes. The storage area for keeping track of signal names is used to point to the value definition for each bit of a signal vector, and to record which primitives define and use a given signal; it required 11.8% of the total storage used. The string space, which stores the text strings used by the other data structures, accounts for 10.6% of the storage space. The "CALL LIST ARRAY" tells which primitives need to be reevaluated when a given bit of a signal is updated, and accounts for 6.9% of the storage space. The "MISCELLANEOUS" category represents a number of minor data structures used within the Timing Verifier, which represent 0.7% of the storage. The Timing Verifier program consists of 4700 lines of PASCAL code requiring 214K bytes of memory when loaded with run-time support.

STORAGE TYPE	K BYTES	% OF STORAGE
CIRCUIT DESCRIPTION	2149	37.8%
SIGNAL VALUES	1843	32.4%
SIGNAL NAMES	680	11.8%
STRINGS SPACE	900	10.6%
CALL LIST ARRAY	389	6.9%
MISCELLANEOUS	41	0.7%
	5684	100.0%

Table 6-2
Storage required by Timing Verifier for 6357 chip example

In general, a compiler which packed its records to take up minimum memory space would permit a significant reduction in storage requirements for the Timing Verifier. Also, additional programming to optimize the data structure for space could result in a non-negligible storage saving. The approach taken for this research was to get a system up and running quickly, in order to test the basic concepts of this approach, and not to attempt to produce an optimized implementation for use in a production environment. Even so, this system has been sufficiently efficient to be used very extensively and highly effectively in the design of the S-1 Mark IIA processor.

7 CONCLUSIONS

The SCALD Timing Verifier has been a very efficient tool for discovering timing errors in the design of large digital systems. It is highly cost-effective from the standpoint of requiring little effort from the designer beyond what is needed to execute the basic design. It is also computationally efficient, allowing a large design to be verified in a relatively small amount of computer time and memory.

Once the timing constraints have been verified, then a simple logic simulator — one which does not have to worry about any timing problems — can find the relatively likely logic errors. The improbable logic errors can then be found either by a hardware simulator or a prototype. The timing in both the prototype and the final production implementation can then be checked with the Timing Verifier.

8 ACKNOWLEDGMENTS

I would like to thank Forest Baskett and Bill vanCieemput for the constant support and guidance they have provided throughout the course of this research. The Fannie and John Hertz Foundation's gracious support has provided me the freedom to pursue this research. Bill Bryson, Mike Farmwald, and Jeff Rubin have been the first users of the SCALD Timing Verifier and their patience and many suggestions for improvement are most appreciated. My thanks also go to the Office of Naval Research and the Naval Electronics System Command for the support to the S-1 Project which has provided the necessary environment for this research, and to Curt Widdoes and Lowell Wood, whose tireless efforts made the formation of the S-1 Project possible.

This work was in part performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract No. W-7405-ENG-48.

9 REFERENCES

- Chicoix, C., Pedoussat, J., and Giambiasi, N., "An Accurate Time Delay Model For Large Digital Network Simulation," Proceedings of the Thirteenth Design Automation Conference, San Francisco, Ca., June 1978, 54-60.
- Harrison, R. A. and Olson, D. J., "Race Analysis of Digital Systems Without Logic Simulation," Proceedings of the Eighth Design Automation Conference, Atlantic City, New Jersey, June 1971, 82-94.
- Krohn, H.E., "Design Verification of Large Scale Scientific Computers," Proceedings of the Fourteenth Design Automation Conference, June 1977, New Orleans, 377-385.
- Kusik, R. and Wesley, P., "Hierarchical Logic Simulation for Digital Systems Development," Proc. Electro/76, Boston, Mass., May 1976, pp. 26.3.1-26.3.8.
- Losleben, P., "Design Validation in Hierarchical Systems," Proceedings of the Twelfth Design Automation Conference, Boston, Mass., June 1975, 431-438.
- McWilliams, T.M. and Widdoes, L.C., "SCALD: Structured Computer-Aided Logic Design," Proceedings of the Fifteenth Design Automation Conference, Las Vegas, Nev., June 1978, 271-277.
- McWilliams, T.M. and Widdoes, L.C., "The SCALD Physical Design Subsystem," Proceedings of the Fifteenth Design Automation Conference, Las Vegas, Nev., June 1978, 278-284.
- Ruehl, A. E., "Electrical Considerations in the Computer Aided Design of Logic Circuit Interconnections," Proceedings of the Tenth Design Automation Conference, June 1973, 262-266.
- S-1 Project Staff, "Advanced Digital Computing Technology Base Development for Navy Applications: The S-1 Project," Prepared for the Naval Systems Division, Office of Naval Research, September 30, 1978. (UCID-18038)
- Szygenda, S.A., "TEGAS2—Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic," Proceedings of the ACM IEEE Design Automation Workshop, June, 1972, 116-127.
- vanCieemput, W.M., "An Hierarchical Language for the Structural Description of Digital Systems," Proceedings of the Fourteenth Design Automation Conference, June 1977, New Orleans, 377-385.
- Wold, M.A., "Design Verification and Performance Analysis," Proceedings of the Fifteenth Design Automation Conference, Las Vegas, Nev., June 1978, 264-270.